研究与开发

演化森林哈希：一种无监督的在线哈希学习算法

寿震宇，钱江波，董一鸿，陈华辉
（宁波大学信息科学与工程学院，浙江 宁波 315211）

摘　要：目前的无监督哈希学习算法在训练阶段需要加载全部的数据，会占据较大的内存空间，并且无法适用于流式数据。探索性地提出了一种无监督在线哈希学习算法——演化森林哈希。针对大规模数据检索场景，通过改进后的演化树学习数据的空间拓扑结构，并提出了路径编码策略将数据点遍历演化树时的路径映射为保相似性二进制编码。为了进一步提高编码查询性能，在演化树哈希的基础上进一步提出在线演化森林哈希，最后在两个被广泛使用的数据集上用实验证明了本文方法的可行性。

关键词：最近邻查询；演化树；在线；哈希学习；集成学习
中图分类号：TP311.3
文献标识码：A
doi: 10.11959/j.issn.1000−0801.2020055

EFH: an online unsupervised hash learning algorithm

SHOU Zhenyu, QIAN Jiangbo, DONG Yihong, CHEN Huahui
Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China

Abstract: Many unsupervised learning to hash algorithm needs to load all data to memory in the training phase, which will occupy a large memory space and cannot be applied to streaming data. An unsupervised online learning to hash algorithm called evolutionary forest hash (EFH) was proposed. In a large-scale data retrieval scenario, the improved evolution tree can be used to learn the spatial topology of the data. A path coding strategy was proposed to map leaf nodes to similarity-preserved binary code. To further improve the querying performance, ensemble learning was combined, and an online evolving forest hashing method was proposed based on the evolving trees. Finally, the feasibility of this method was proved by experiments on two widely used data sets.

Key words: nearest neighbor query, evolving tree, online, hash learning, ensemble learning

收稿日期：2019−09−29；**修回日期**：2020−02−26
通信作者：钱江波，qianjiangbo@nbu.edu.cn
基金项目：浙江省自然科学基金资助项目（No.LZ20F020001，No.LY20F020009）；国家自然科学基金资助项目（No.61472194，No.61572266）；宁波市自然科学基金资助项目（No.2019A610085）
Foundations Items: Zhejiang Provincial Natural Science Foundation of China (No.LZ20F020001, No.LY20F020009), The National Natural Science Foundation of China (No.61472194, No.61572266), Ningbo Municipal Natural Science Foundation of China (No.2019A610085)

1 引言

随着互联网以及各类电子设备的快速发展，各类数据，例如文本、图像和视频正在飞速增长。在很多应用场景下，人们都需要从大规模数据中检索相关内容。然而，在大规模数据中，查找给定查询点的精确最近邻所花费的计算时间是让人无法接受的。为了解决这个问题，最近有大量的研究已经致力于相似最近邻（approximate nearest neighbor，ANN）搜索，在大规模数据中，ANN 检索的效果可以代替精确最近邻检索，而且速度非常快[1-4]。基于哈希学习的 ANN 检索是众多 ANN 检索技术中较为知名的一种，它结合机器学习的机制将数据点映射到汉明空间，用汉明距离代替原始数据的欧氏距离，在保证准确率的同时，大幅度地减少检索时间和存储代价。近些年来，涌现出了许多优秀的哈希学习算法，根据学习模型是否利用样本的标签信息，可以分为无监督模型和监督模型。考虑到获取标签信息需要巨大的人工成本，因此无监督哈希学习算法得到了更广泛的应用，然而目前的无监督哈希学习算法在训练阶段需要加载全部的数据，会占据大量的内存空间，并且无法适用于流式数据。本文探索性地提出了一种无监督在线哈希学习算法——演化森林哈希（evolving forest hash，EFH）。针对大规模数据检索场景，通过改进后的演化树学习数据的空间拓扑结构，并提出了路径编码策略，将数据点遍历演化树时的路径映射为保相似性二进制编码。

2 相关工作

一般来讲，目前主流的哈希算法分为两类：数据独立哈希和数据依赖哈希。在数据独立哈希中，哈希函数族的生成独立于数据集，其典型代表为局部敏感哈希（locality sensitive hashing，LSH）[6-10]，它利用一组随机的哈希函数来建立哈希表，使得相似的数据点能够以较大的概率被映射到相似的哈希桶中，但其缺点是索引的建立过程是数据独立的，在实际大规模数据集的检索中效果较差。数据依赖哈希又称为哈希学习[5]，通过机器学习机制将数据映射为保相似性的二进制编码，是机器学习技术在数据检索领域的一个典型应用，哈希学习最重要的目的是实现哈希编码的保相似性，具体来说，在原始空间中距离较小的两个数据点在被映射到汉明空间之后，仍然能保持较小的汉明距离，对于距离远的数据点，在被映射之后，汉明距离仍然较大。近些年来，许多哈希学习算法相继被提出，根据学习模型是否利用样本的标签信息，可以分为无监督哈希算法和带监督哈希算法。无监督哈希算法的著名代表有主成分哈希（principal component analysis hashing，PCAH）[11]、迭代量化（iterative quantization，ITQ）[28]、*K* 均值哈希（*K*-means hash，*K*MH）[27]等，其中 PCAH 使用主成分分析将输入数据空间投影到低维空间中，再将低维数据映射为哈希编码，ITQ 试图寻找一种对原始数据最优的旋转方式，在将原始数据映射为二进制编码时量化损失最小，KMH 从聚类的角度设计哈希编码，基本思想是，将数据聚成 *K* 类，类内数据采用矢量量化策略，统一量化为聚类中心点的值，此外，根据保相似性原则对每个聚类中心点进行编码。在查询阶段，将数据点 *x*、*y* 之间的距离近似为对应聚类中心的哈希码的汉明距离。带监督哈希算法主要包括 RBM[12]、BRE[13]、MFH[14]、IMH[15]、MLH[16]，虽然监督哈希显示出比无监督哈希法更高的搜索准确性，但是，它们的训练都需要标签信息，在海量数据时代，数据规模大、更新速度快，数据标签的获取常常需要巨大的人工成本，因此无监督哈希在实际应用更有意义。然而绝大多数的无监督哈希算法需要一次性加载所有数据，会占用大量的内存，无法适用于流式数据，并且相关研究较少。

3　基础知识

下面将介绍与本文研究工作密切相关的矢量量化（vector quantization，VQ）[17]、自组织映射（self-organizing map，SOM）[18-19]以及 SOM 的变体——演化树（evolving tree，E-tree）[20]，同时，E-tree 也是本文研究工作的基础模型。

3.1　矢量量化

VQ 是一种经典的有损数据压缩技术，它通过原型向量的分布来模拟输入数据的概率密度。VQ 将原始数据空间划分为多个区域，每个区域中都使用某个原型向量来表示该区域中的所有数据点，从而实现对原始实数向量的量化。用于表示数据点的原型向量称为码字，所有原型向量构成的集合称为码本 C。一旦确定了码本，每个输入数据点 x_i 可以用码本中与之最接近的码字 c_i 表示，由此产生的误差即矢量量化的损失为：

$$E=\sum_{i=1,c_k\in C}^{n}\|x_i-c_k\| \tag{1}$$

其中，C 表示码本，c_k 表示码字，k 为码字的索引。

$$k=\arg\min_i(\|x-c_i\|) \tag{2}$$

3.2　演化树

SOM 是一种在数据分析任务中被广泛使用的神经网络模型，本质上，SOM 也是一种矢量量化，但与其他量化方法不同的是，它能够从全局空间中选择最优的码字，并且在训练的过程中，能够保留原始数据的空间拓扑结构。SOM 的量化损失为：

$$E=\frac{1}{n}\sum_{c_k\in C}\sum_{x_i\in X}\|x_i-c_k\|N(i,k) \tag{3}$$

其中，$N(i,k)$ 为邻域函数，在多数的 SOM 研究工作中，都使用高斯领域函数，即：

$$N(i,k)=\exp\left(\frac{\left\|r_k-r_i\right\|^2}{2\sigma^2(t)}\right) \tag{4}$$

在绝大多数操作开始前，SOM 必须要先从所有的网格节点中找到最佳匹配单元（best match unit，BMU），这项操作的复杂程度取决于网络的大小，在大数据分析中，需要消耗大量的时间[21-23]。

E-tree 是一种新型的 SOM 网络，从聚类层面上看，E-tree 是一种快速的层次聚类方法，适用于大型的高维问题；从索引层面上来看，E-tree 可以看作向量数据的索引，类似于 R 树等其他算法。演化树的层次性和近似性可以帮助它处理其他分类方法无法处理的大规模数据近似查询问题。

类似于 SOM，E-tree 的每个节点都拥有自己的原型向量 w，在训练的过程中，每个节点都会记录自己成为 BMU 的次数，当 BMU 的次数达到某个预设值之后，节点分裂。演化树在初始状态时，只有 1 个节点，也就是根节点，训练根节点的方式与训练 1×1 的 SOM 网络一致，直到根节点成为 BMU 的次数达到某个预设值之后，根节点开始分裂。具体操作是：创建 n 个新节点，并将这些新节点标记为根节点的子节点，属性为叶子节点，这时，根节点已经成为一棵拥有 n 个叶子节点、1 个躯干节点的 E-tree，接下来所有的操作只会在于叶子节点上进行，也就是说，当 1 个叶子节点成为躯干节点之后，该节点的所有信息将不会更新。这么做，一方面是因为躯干节点的任务仅仅是保持与其他躯干节点和叶子节点之间的联系；另一方面是为了让 E-tree 能快速、稳定地收敛。

当新的数据点 x 到来时，寻找 x 对应的 BMU 是一个自上而下的过程，首先从根节点开始，计算每个孩子节点的原型向量 w 与 x 之间的欧氏距离，得到与 x 的欧氏距离最小的 w 的孩子节点，如果该孩子节点为叶子节点，则该孩子节点即要找的 BMU，如果该孩子节点为躯干节点，则重复之前的操作，直到找个 1 个叶子节点作为 BMU。当找到 BMU 时，需要调整 BMU 及其邻域节点的

原型向量w，调整的方法类似于传统的 SOM 网络，利用式（5）进行更新：

$$w_i(t+1)=w_i(t)+N(i,k)\cdot(x(t)-w_i(t)) \quad (5)$$

由于传统的 SOM 网络是对称、静态的矩形结构，BMU 与其邻域节点之间的欧氏距离可以直接用两者网格坐标进行计算，但是，E-tree 并不具有一般 SOM 具有的性质，并不能用传统的方式计算邻域距离，E-tree 采用 BMU 到其邻域节点的跳数作为邻域距离。

4 演化树哈希

E-tree 的提出为大规模数据分析任务提供了一个高效的工具，如何通过 E-tree 得到保相似性的哈希编码是本文的主要工作。首先介绍将原始 E-tree 用于大规模数据时存在的缺陷以及改进方式，接下来提出了演化树哈希（evolving tree hashing，ETH），并详细介绍如何利用改进后的 E-tree 得到保相似性的哈希编码。

4.1 演化树改进

在海量高维数据的背景下，如果利用原始 E-tree 实现哈希近邻检索，E-tree 存在以下不足。首先，E-tree 把整棵树中所有的叶子节点都当成 BMU 的领域，在网格型 SOM 中，节点的位置是静态的、有规则的。网格型 SOM 在调整领域节点时，只需简单的线性扫描就能调整所有的领域节点。然而，在 E-tree 的生长过程中，叶子节点是在不断变化的，在达到预设的 BMU 次数后，叶子节点会分裂，生长出新的叶子节点，而原节点则会变为躯干节点。如果想要在训练的过程中快速找到所有的叶子节点以及计算 BMU 与叶子节点的距离，则需要额外的数据结构存储叶子节点，并实时更新这些数据结构，这会造成极大的时空开销。其次，在 E-tree 的训练过程中，邻域调整仅在树不平衡的状态下有较大的意义，在平衡状态下，会形成大量的冗余操作。演化树演变过程如图 1 所示。如图 1（e）所示，假设 A_1 为当前

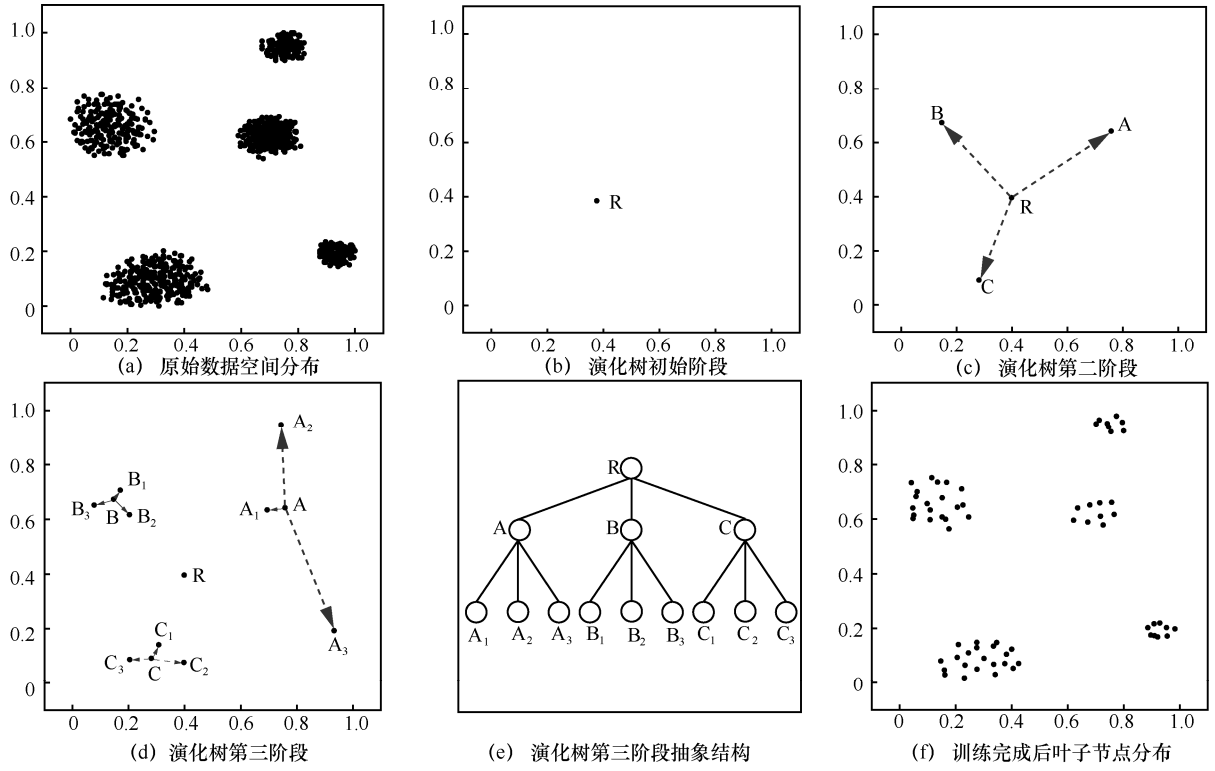


图 1 演化树演变过程

BMU，这时需要对 BMU 和 BMU 的邻域节点做调整，将 A_1 的邻域分为两部分：$n_1=\{B_1,B_2,B_3,C_1,C_2,C_3\}$ 与 $n_2=\{A_2,A_3\}$。首先调整 n_1 内的节点，按照原始 E-tree 邻域距离的计算方式，可以发现 A1 与 n_1 的节点的跳数皆为 3，这意味着，无论 n_1 内有多少个叶子节点，对 n_1 内节点的调整力度都是一致的，调整操作过于冗余。同理，在调整 n_2 时亦如此。在将 E-tree 用于哈希编码时，为了最大化比特位的利用率，需要使演化树尽可能地平衡，因此，在演化树哈希中，本文将邻域函数简化为式（6），即每次更新只对 BMU 做调整。

$$N(i,k)=\begin{cases}1, & i=k\\ 0, & \text{其他}\end{cases} \tag{6}$$

算法 1　训练演化树

输入　D（数据集），Round（迭代次数），max_depth（树的最大深度），child_number（孩子节点数量），max_bmu_times（bmu 限制次数），beta（扰动系数），learning_rate（学习率）

输出　演化树根节点

初始化　随机生成一个根节点

for r from 1 to Round：

　　for a coming point in Data：

　　　　bmu = find_bmu（point）

　　　　if bmu.bmu_times < max_bmu_times：

　　　　　　Update bmu.w by 式（5）

　　　　else if bmucurrentdepth < max_depth：

　　　　　　creat child_number nodes arrcording to beta

return root node

此外，为了保证 E-tree 能够稳定收敛，本文引入了权重继承机制，即当叶子节点分裂出新节点时，新节点会继承父节点的大部分权重，并且加入小部分的随机扰动：

$$w'=(1-\beta)w+\beta r \tag{7}$$

其中，w' 为新节点的权值向量，w 为父节点的权值向量，r 是与 w 相同维度的随机单位向量，β 为超参数，用于控制随机扰动程度。

本文对简化后的 E-tree 训练过程及效果做了可视化分析，可视化实验参数为：Rounds=10、max_bmu_times=60、learning_rate=0.1、β=0.05。如图 1（e）所示，训练数据由 890 个二维坐标点构成，在二维空间中组成 5 个类簇。图 1（b）~图 1（d）分别表示 E-tree 训练过程中的前 3 个阶段，其中带箭头的虚线表示当前阶段叶子节点的生长方向。在初始阶段，空间只有一个根节点 R。随着训练进行，在第二阶段 R 分裂出 3 个叶子节点 A、B、C。在第三阶段，A、B、C 3 点分裂出各自的子节点，{A_1、A_2、A_3}、{B_1、B_2、B_3}、{C_1、C_2、C_3}，根据以上叶子节点的坐标值可以发现，此时，E-tree 已经简略地学习到了数据的拓扑结构。图 1（f）表示 E-tree 完成训练后所有叶子节点的空间位置，可以发现此时 E-tree 已经学习到了训练数据的拓扑结构。

4.2　演化树哈希编码

演化树可以认为是一种在线聚类算法，基于聚类的哈希学习算法的损失函数包括两个部分：第一部分为量化损失，即将空间中的数据点量化为对应的码字所产生的损失，第二部分为保相似性损失。

首先考虑量化损失，根据式（1）可知，矢量量化的损失大小取决于码字数量以及分布是否能够近似地模拟原始数据的分布，而演化树的训练过程，实质上是不断优化式（1）的过程，从图 1（f）中可以看到，演化树能较准确地学习到原始数据的分布，这就相当于得到了式（1）的较优解，因此，将训练完成后的演化树作为映射函数 f_{tree}，叶子节点的集合作为矢量量化的码本，对于 d 维的向量 $x\in R^d$，$f_{\text{tree}}(x)$ 表示 x 对应的叶子节点。根据矢量量化，本文将任意两个数据点 x_i、x_j 在原始空间中的距离近似为它们对应的 BMU 距离，即对应的叶子节点的距离：

$$d(x_i,x_j)\simeq d(c_i,c_j)= d(f_{\text{tree}}(x_i),f_{\text{tree}}(x_j))=d(\text{BMU}_i,\text{BMU}_j) \quad (8)$$

其中，$d(x)$表示欧氏距离。

第二步，为叶子节点设计保相似性的二进制编码，根据哈希学习的保相似原则，保相似性损失函数如式（9）所示：

$$\min_{b(\cdot)} E = \min_{b(\cdot)} \sum_{c_i\in C}\sum_{c_j\in C}(d(c_i,c_j)-\lambda d_h(b(c_i),b(c_j)))^2 \quad (9)$$

其中，C 为叶子节点集合，即码本；$b(c)$ 表示 c 对应的汉明码；$d_h(\cdot)$ 表示汉明距离；λ 是平衡两部分损失的超参数，λ 的不同取值对实验结果有较明显的影响，本文将在实验部分对 λ 进行详细的讨论。在式（9）中，假设使用 n 位汉明码为所有的码字编码，由于 C 已经确定，因此只能不断更新 $b(c)$ 才能对式（9）进行优化，然而汉明码是离散的，只有穷举编码的全排列才能得到最优的 $b(c)$，这样的方式算法复杂度为 $O(2^n!)$，无法被现实任务采用。

为了设计保相似性的汉明码，本文提出路径编码策略对式（9）进行优化。基本思想是，每个码字 c_i 在演化树中都有唯一的路径，对 c_i 编码等价于对 c_i 对应的路径进行编码，因此，式（9）等价于：

$$\min_{b(\cdot)} E = \min_{b(\cdot)} \sum_{p_i\in P}\sum_{p_j\in P}(d_p(p_i,p_j)-\lambda d_h(b(p_i),b(p_j)))^2 \quad (10)$$

其中，P 表示根节点到所有叶子节点的路径的集合，d_p 是抽象函数，表示 p_i、p_j 之间的距离，d_h 为汉明距离，$b(p)$ 表示路径 p 对应的汉明码，式（10）的基本含义为，找到一个最优的 $b(\cdot)$，能够使相似的路径在映射为哈希编码之后，它们的汉明距离能够尽可能地小，不相似的路径在映射为哈希编码之后，之间的汉明距离能够尽可能地大。

由于 d_p 是表示两条路径差异程度的抽象函数，并没有具体的度量形式和数学表达式，所以无法对式（10）直接优化求解，根据演化树的训练方式和分裂原则容易得出结论，在树中，距离相近的叶子节点在树中的路径也是相似的，由此可以采用贪心策略逐步优化式（10），即每一个躯干节点都对子节点进行编码，实现子节点之间的局部保相似性，所以优化目标成为：

$$F(W)=\sum_{w_i\in W}\sum_{w_j\in W}(d(w_i,w_j)-\lambda d_h(b(w_i),b(w_j)))^2 \quad (11)$$

$$\min E=\sum_{W_i\in N}\min_{b(\cdot)}F(W_i) \quad (12)$$

其中，$b(\cdot)$、d_h 与式（10）中的含义相同，$w\in R^d$ 表示节点的权值向量，$W_i=\{w_1,w_2,\cdots,w_c\}$ 表示躯干节点 i 所有孩子节点的权重向量，$N=\{W_1,W_2,\cdots,W_n\}$，λ 为超参数，在实验中取值为 0.6 时效果最佳。

优化每一个 $F(W_i)$ 都是独立的，且操作一致，可以设计一个所有躯干节点共用的局部汉明码本 M_{local}，为了严格保证子节点之间的局部保相似性，M_{local} 须要符合以下两个要求：汉明码的个数等于躯干节点的子节点数量 n，并且汉明码互不相同；汉明码之间的汉明距离范围是 $[1,n-1]$，且任何一个汉明码与其他各个汉明码的距离都不相同，以 $n=3$ 为例，$M_{\text{local}}=\{00,11,01\}$。由于孩子节点的数量并不会明显影响演化树学习原始数据拓扑结构的效果，因此，为了降低编码部分的算法复杂度，子节点的数量一般设为 3 或 4，此时 M_{local} 全排列 I 的数量仅仅是 6 或者 24。通过遍历 I 求解最优的局部编码，编码部分算法的时间复杂度为 $O(6n)$ 或 $O(24n)$。

在完成对所有的子节点编码之后，只需要记录数据点 x_i 对应的叶子节点与根节点之间的路径便可以得到数据点的编码。根据路径中每个节点的编码，数据点 x_i 映射为路径节点编码的有序组

合 $y_i = u_1 u_2 \cdots u_j \cdots u_{\text{dep}}$，其中 u_j 是路径中的第 j 个节点的编码，dep 代表树的最大深度。演化树哈希编码方式如图 2 所示，$y_{A_1} = 1111$。由于演化树并不是平衡树，如果 x_i 最后所在的叶子的深度小于树的最大深度，为了保证编码长度的统一，利用当前叶子节点的编码补全后续缺失的编码，如图 2 中的点 C，$y_c = 00$，编码补全之后，$y_{c_1} = 0000$。

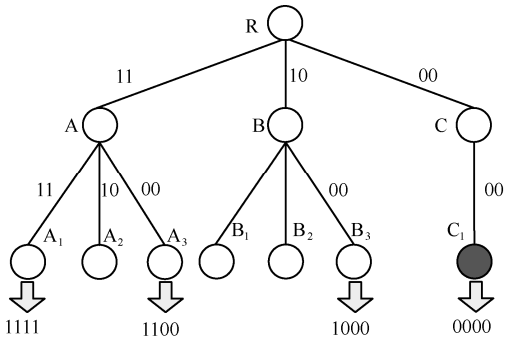


图 2　演化树哈希编码方式

5　演化森林哈希

5.1　演化树哈希局限性

在上述的演化树哈希中，本文采用贪心的路径编码策略将训练得到的叶子节点转化为保相似性的二进制编码。然而，演化树哈希的适用范围仅限于短编码，但短编码的查询性能往往比长编码差，难以胜任实际任务，想要通过演化树哈希获得更高查询性能的长编码是非常困难的。从时空开销的角度分析演化树哈希的编码长度与树的深度成正比，单纯的通过增加树的深度来扩大编码是不现实的。随着树深度的增加，整棵树的节点数量 n 呈指数增长，$n = 1 - m^{\text{dep}} / 1 - m$，其中 m 为孩子节点的数量。以孩子节点数量为 3 的演化树为例，当所需要的编码长度为 64 位时，演化树的深度为 32，则点数量 $n \approx 3^{32}$，这时节点规模已经达到了千万亿级别，在有限内存资源和有效的时间范围内几乎不可能完成演化树的训练以及路径编码，此外叶子节点的数量已经远远超过了数据点的数量，这样的量化方式没有任何意义。

另一方面，树深度的增加并不能显著提升编码的查询性能。当树的深度超过某个阈值时，编码的查询性能会逐步下降，假设集合 N_i 表示 x_i 的近邻 $N_i = \{x_j \mid \|x_i - x_j\| \leqslant \varepsilon\}$，当 N_i 中的数据点都能落在叶子节点 L 上时，根据路径编码，这些数据点的哈希编码之间的汉明距离皆为 0，很明显，当 L 再次分裂时，这些数据点的哈希编码之间的汉明距离必然大于 0，而且随着树的加深，误差会越来越大。此外，演化树是非平衡树，数据集在原始空间中，并不属于均匀分布，数据较为密集的空间区域中会有较多的叶子节点，较为稀疏的区域则存在较少的叶子节点，因此树深度的增加会加剧演化树的不平衡性，编码哈希中，需要大量编码补全操作来统一编码长度，这会使得哈希编码过度冗余。

算法 2　训练在线演化森林

输入　D（数据集），Round（迭代次数），T（森林中树的数量）。

输出　演化树根节点集合。

初始化　随机创建 T 个根节点

for r in Round：

　for 新到来的数据点：

　　for t from 1 to T：

　　$K \leftarrow$ Possion（1）

　　使用该数据点调整第 t 棵树 K 次

return 根节点集合

演化树哈希在编码方式上存在一定的缺陷，限制了编码性能。基于贪心策略的路径编码能够让相近的数据点实现非常好的保相似性，例如图 2 中的 A_1、A_2 和 A_3。但是对于距离较远的数据点可能存在较大的误差，例如图 2 中 A_3 与 B_3 的汉明距离小于 A_1 与 A_3 之间的汉明距离，但实际上，A_1 与 A_3 之间的欧氏距离更小。另外，在演化树哈希的训练阶段采用贪心的方式寻找 BMU 时，由于贪心算法只能得到局部最优解，因此不一定能找到距离训练数据点最近的叶子节点，在调整

BMU 时，可能会存在一定误差，而且随着树深度的增加，这种误差的可能性会越大。

5.2 构建演化森林

根据第 5.1 节可知，演化树哈希的编码长度以及查询性能非常有限，而且通过增加树的深度实现纵向扩展编码并不可行，因此如何有效地扩展编码从而提升编码的查询性能成为本文亟需解决的关键问题。本文从机器学习任务的角度重新看待相似最近邻查询问题，基于哈希技术的相似最近邻搜索可以概括为：对于给定的查询点 q_i，通过哈希模型映射为汉明码 y_i，再计算 y_i 与候选集（提前转化为汉明码）之间的汉明距离，汉明距离越小，越有可能是最近邻。因此，可以将整个 ANN 检索问题转化为一个以汉明距离为分类依据，以近邻与非近邻为标签的广义二分类问题，由于多种条件的限制，演化树哈希模型等价于一个弱学习器。在机器学习领域，往往可以通过集成学习，将多个弱学习器组合为性能优异的强学习器。其中，Bagging 是并行式集成学习方法最著名的代表之一[31]，Bagging 的弱学习器之间没有依赖关系，可以并行生成，为了实现编码的简洁与高效性，本文采用 Bagging 扩展编码。同时考虑到传统的 Bagging 需要得到全部的样本数据，并不适用于流式数据采样，本文最终采用 Online-Bagging[28]。

基于 Online-Bagging，本文将演化树哈希作为弱学习器，在此基础上提出在线演化森林哈希，对于森林中每一棵树，从强度为 1 的泊松分布中随机采样一个数字，记为 K，然后使用数据流中的当前数据点 x_i 训练演化树 K 次。另外，森林的编码方式本质上与单树编码的方式是一致的，森林中的每棵树都是独立的，把每棵树的编码进行有序组合后就是森林编码，即通过森林把数据点 x_i 映射为二进制编码：$y_i=[y_i^{(1)},y_i^{(2)},\cdots,y_i^{(k)},\cdots,y_i^{(T)}]$，其中 $y_i^{(k)}$ 表示森林中第 k 棵树对 x_i 的编码，T 表示树的总数，演化森林哈希编码方式如图 3 所示。

表面上只是通过 Online-Bagging 引入了样本的随机性，实质上，由于演化树的训练方式为在线训练，不同的随机子样本集合在本质上引入了另外一种随机性——树生长方向的随机性，当演化树扩展为演化森林之后，可以从随机从原始空间中的多个位置出发捕捉数据的空间分布，树越多越能全面地捕捉数据的空间拓扑结构，从而缓解路径编码的缺陷。

6 实验结果

6.1 数据集选取

本文中的算法使用 Python 3.6 编程实现，运行机器配置 CPU 为 Intel Core i5-4590 CPU @ 3.30 GHz，内存 16 GB，分别在 CIFAR10、GIST1M 这两个

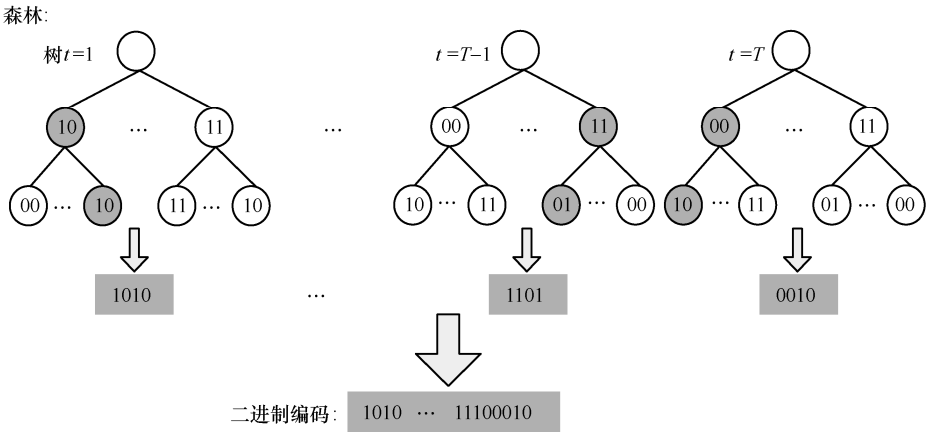


图 3 演化森林散列编码示意图

公开数据集上与其他经典的哈希方法进行了对比，数据集简介如下。

- CIFAR10 数据集包含 60 000 个 32×32 大小的彩色图像，分属于 10 个类别，每个类别包含 6 000 张图片。在本次实验中，对 CIFAR10 数据集中的每张照片提取出 512 维的 GIST 特征，并从中随机挑选出 1 000 张图片作为查询集，其余 59 000 张作为训练数据。
- GIST1M 数据集包含 1 000 000 个 960 维的图像的全局 GIST 特征集。在本次实验中，随机抽取 1 000 个样本作为查询集，其余为训练数据。

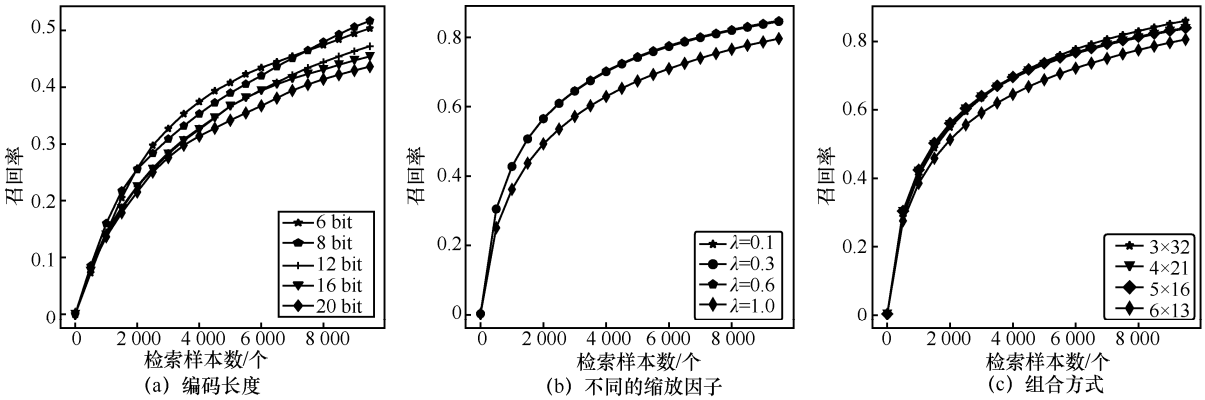
为模拟流数据形式，在实验过程中将 CIFAR10、GIST1M 训练集分成若干块。

6.2　评价指标

本文使用召回率（recall）作为评价指标。对于给定的数据集合进行最近邻检索，真实类别和检索结果存在 4 种关系：检索到的近邻数（true positive，TP）、检索到的非近邻数据（false positive，FP）、未检索到的近邻数据（false negative，FN）、未检索到的非近邻数据（true negative，TN），召回率的计算方式为 TP/（TP+FN），对于查询点 q_i，定义训练集中与 q_i 欧氏距离最近的前 K 个点作为 q_i 的近邻，在本次实验中 K 取 100。

6.3　实验比较与分析

在本次实验中，采用孩子节点数量为 3 的演化树作为基础模型，图 4 中的实验结果均在 CIFAR10 数据集上产生。图 4（a）展示了由演化树哈希得到的 5 种编码长度下的召回率，长度分别为 6 bit、8 bit、12 bit、16 bit、20 bit，演化树的参数为：Round=10，learning_rate=0.1，max_bmu_times=6 000，β=0.9，最大深度分别为 4、5、7、9、11。从图 4（a）中可以看出，长度为 8 bit 的召回率在整体上略优于 6 bit，但是随着编码长度的增加，召回率却在逐步下降，当编码长度为 20 bit 时，效果最差，通过增加树的深度来扩展编码并不可行，实验结果与上文中对演化树哈希的性能分析是一致的。图 4（b）展示了在路径编码阶段，不同 l 对召回率的影响，可以看出，$\lambda \in [0.1,0.6]$ 时能有较高的召回率，并且较稳定，因此在后续的试验中，λ 统一设置为 0.6。演化森林的编码方式比较灵活，例如，想要得到长度约 128 bit 的编码，可以由树的最大深度为 5、树的数量为 16（简记为 5×16）的森林产生，也可以由 3×32、4×21、6×13 等方式产生，图 4（c）展示了由 3×32、4×21、5×16、6×13 得到的约 128 bit 编码的召回率，由图 4（c）可以看出，3×32 的效果最佳，但由不同组合方式得到的编码的召回率曲线都比较接近，这体现了本文算法的稳定



(a) 编码长度　(b) 不同的缩放因子　(c) 组合方式

图 4　演化树、森林参数讨论

性，同时，在实际应用时，本文也推荐使用较小的树深度和较多的树来实现编码以获得更高的召回率。

接下来，本文将 EFH 算法与其他两种经典查询算法进行对比以验证算法的有效性。3 种算法分别为 EFH、SKLSH[26]以及 PCAH，其中，EFH、PCAH 属于学习型哈希，SKLSH 属于数据独立的哈希算法。图 5 为在 GIST1M 数据集下，二进制编码长度分别为 32、64 以及 128 时，3 种算法的召回率曲线。从图 5 中可以看到，3 种检索算法中，EFH 在整体上效果最好。当编码长度为 32 bit 时，EFH 中树的数量较少，仅 4 棵，模型很有可能处于欠拟合状态，因此效果不如 PCAH，但优于 SKLSH。随着编码长度的增加，EFH 的召回率得到大幅度的提升，效果最佳，PCAH 的召回率并不能随着编码维度的增加而提升。SKLSH 作为一种数据不依赖型哈希算法，在编码长度较短时，其查询效果比其他 3 种算法差，随着编码维度的增加，SKLSH 的增长效果也比较明显，但是二进制编码维度的增加势必会带来算法的代价增加，并且算法的查询性能变高的趋势会随着编码长度的增加越来越缓慢，因此，需要选择适当的编码长度使得在付出最小代价的同时获得最大的收益。

图 6 为在 GIST 数据集下，二进制编码长度分别为 32、64 以及 128 时，3 种算法的召回率曲线。从图 6 中可以看出，曲线走向与在数据集 CIFAR 下的情况类似，整体上 EFH 依然是效果最为优越的算法。

7 结束语

本文首先提出了演化树哈希，进一步地将集

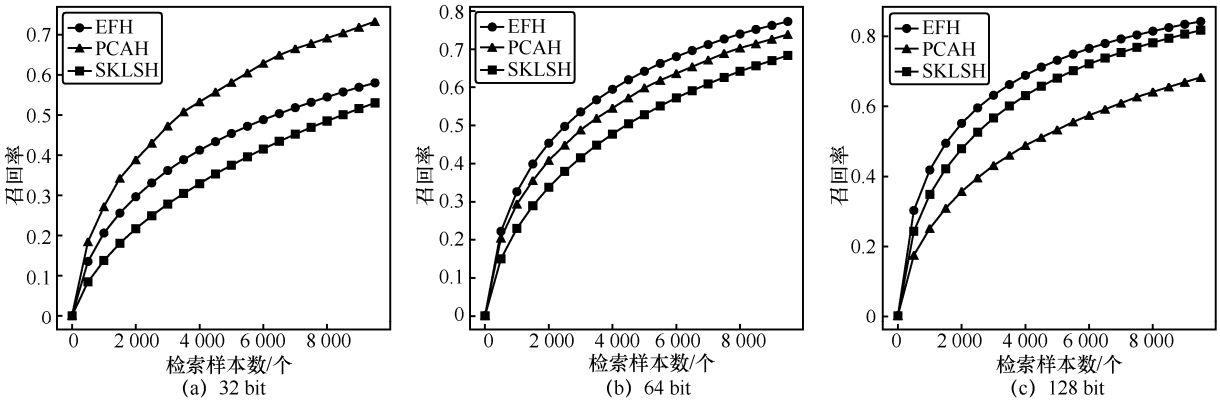


图 5 CIFAR 数据集下不同比特的召回率曲线

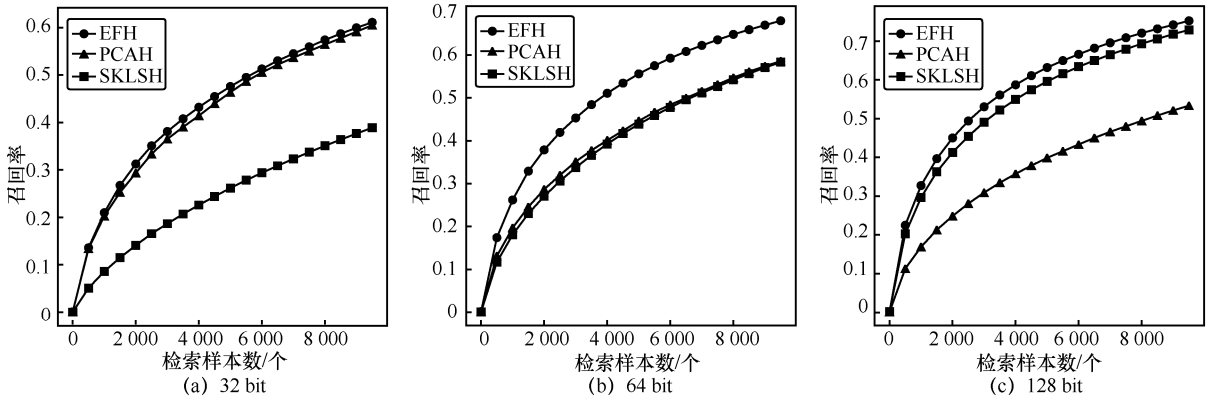


图 6 GIST 数据集下不同比特召回率曲线

成学习引入哈希学习中，在此基础上提出了一种新的哈希模型，即在线的演化森林哈希，通过调整树的深度与树的数量可以灵活地调整编码长度。随着海量数据时代的到来，单机环境已经不再适合处理大规模的数据，目前已经出现一些较成熟的分布式平台[30]，由于演化森林哈希中的树都是相互独立的，非常适合分布式平台，如何在分布式上实现演化森林哈希，是将来很有意义的工作。

参考文献：

[1] ATKINSON P, ORLOWSKA M. Similarity search in high dimensions via hashing[C]//Proceedings of International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann, 1999: 518-529.

[2] ANDONI A, INDYK P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions[J]. Communications of the ACM, 2008, 51(1): 117-122.

[3] ATHITSOS V, POTAMIAS M, PAPAPETROU P, et al. Nearest neighbor retrieval using distance-based hashing[C]//Proceedings of International Conference on Data Engineering. Piscataway: IEEE Press, 2008: 327-336.

[4] ZONG Z. Exploiting Web images for semantic video indexing via robust sample-specific loss[J]. IEEE Transactions on Multimedia, 2014, 16(6): 1677-1689.

[5] WANG J, ZHANG T, SONG J, et al. A survey on learning to hash[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, 40(4): 769-790.

[6] DATAR M, IMMORLICA N, INDYK P, et al. Locality-sensitive hashing scheme based on P-stable distributions[C]//Proceedings of ACM Symposium on Computational Geometry. New York: ACM Press, 2004: 253-262.

[7] SHRIVASTAVA A, LI P. Asymmetric LSH (ALSH) for sublinear time maximum inner product search [J]. Advances in Neural Information Processing Systems, 2014(3): 2321-2329.

[8] ANDONI A, INDYK P, LAARHOVEN T, et al. Practical and optimal LSH for angular distance[C]//Proceedings of Neural Information Processing Systems. Piscataway: IEEE Press, 2015: 1225-1233.

[9] QIAN J, ZHU Q, CHEN H. Multi-granularity locality-sensitive bloom filter[J]. IEEE Transactions on Computers, 2015, 64(12): 3500-3514.

[10] HUANG Q, FENG J, ZHANG Y, et al. Query-aware locality-sensitive hashing for approximate nearest neighbor search[J]. Proceedings of the VLDB Endowment, 2015, 9(1): 1-12.

[11] GONG Y, LAZEBNIK S, et al. Iterative quantization: a procrustean approach to learning binary codes[C]//Proceedings of Computer Vision and Pattern Recognition. Piscataway: IEEE Press, 2011: 817-824.

[12] SALAKHUTDINOV R, HINTON G. Semantic hashing[J]. International Journal of Approximate Reasoning, 2009, 50(7): 969-978.

[13] KULIS B, DARRELL T. Learning to hash with binary reconstructive embeddings[C]//Proceedings of Annual Conference on Neural Information Processing Systems. Redhook: Curran Associates, 2009: 1042-1050.

[14] KONG W, LI W, GUO M, et al. Manhattan hashing for large-scale image retrieval[C]//Proceedings of International Conference on Research and Development in Information Retrieval. New York: ACM Press, 2012: 45-54.

[15] SONG J, YANG Y, YANG Y, et al. Inter-media hashing for large-scale retrieval from heterogeneous data sources[C]//Proceedings of International Conference on Management of Data. New York: ACM Press, 2013: 785-796.

[16] NOROUZI M, FLEET D. Minimal loss hashing for compact binary codes[C]//Proceedings of International Conference on Machine Learning. Madison: Omni Press, 2008: 353-360.

[17] LI J, TIAN D, ALREGIB G. Vector quantization in multiresolution mesh compression[J]. IEEE Signal Processing Letters, 2006, 13(10): 616-619.

[18] KOHONEN T. The self-organizing map[J]. Neurocomputing, 1998, 21(1): 1-6.

[19] VESANTO J, ALHONIEMI E. Clustering of the self-organizing map[J]. IEEE Transactions on Neural Networks, 2000, 11(3): 586-600.

[20] PAKKANEN J, IIVARINENJ, OJA E. The evolving tree—analysis and applications[J]. IEEE Transactions on Neural Networks, 2006, 17(3): 591-603.

[21] BLACKMORE J, MIIKKULAINEN R. Visualizing high-dimensional structure with the incremental grid growing neural network[C]//International Conference on Machine Learning. San Francisco: Morgan Kaufmann, 1995: 55-63.

[22] FRITZKE B. Growing cell structures: self-organizing network for unsupervised and supervised learning[J]. Neural Networks, 1994, 7(9): 1441-1460.

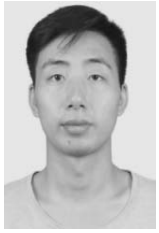
[23] KOIKKALAINEN P, OJA E. Self-organizing hierarchical feature maps[C]//International Joint Conference on Neural Network. Piscataway: IEEE Press, 1990: 279-284.

[24] LIU Y, CUI J, HUANG Z, et al. SK-LSH: an afficient index structure for approximate nearest neighbor search[J]. Proceedings of the VLDB Endowment, 2014, 7(9): 745-756.

[25] ZHOU Z H. Machine learning[M]. Beijing: Tsinghua University Press, 2016.

[26] OZA N C, RUSSELL S. Online bagging and boosting[C]// International Conference on Systems. Piscataway: IEEE Press, 2005: 2340-2345.
[27] HE K, WEN F, SUN J. *K*-means hashing: an affinity-preserving quantization method for learning binary compact codes[C]// Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE Press, 2013: 2938-2945.
[28] GONG Y, LAZEBNIK S, GORDO A, et al. Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval[J]. IEEE Transactions on Pattern Analysis Machine Intelligence, 2013(35): 2916-2929.
[29] BREIMAN L. Bagging predictors[J]. Machine Learning, 1996, 24(2): 123-140.
[30] SHARMIN N, LU X. Performance characterization and acceleration of in-memory file systems for Hadoop and Spark applications on HPC clusters[C]//Proceedings of IEEE International Conference on Big Data, Big Data. Piscataway: IEEE Press, 2015: 243-252.
[31] 郭佳睿, 魏进武, 张云勇. 大数据助力运营商提升规模化运营核心力策略[J]. 电信科学, 2018, 34(1): 120-125.
GUO J R, WEI J W, ZHANG Y Y. Strategies for enhancing core capability of large-scale operation for national telecom operators assisted by big data[J]. Telecommunications Science, 2018, 34(1): 120-125.
[32] 陈涛, 鲁萌, 陈彦名. 运营商大数据技术应用研究[J]. 电信科学, 2017, 33(1): 130-134.
CHEN T, LU M, CHEN Y M. Research on operators' big data technologies and applications[J]. Telecommunications Science, 2017, 33(1): 130-134.
[33] 郭佳睿, 魏进武, 张云勇. 大数据助力运营商提升规模化运营核心力策略[J]. 电信科学, 2018, 34(1): 120-125.
GUO J R, WEI J W, ZHANG Y Y. Strategies for enhancing core capability of large-scale operation for national telecom operators assisted by big data[J]. Telecommunications Science, 2018, 34(1): 120-125.

[作者简介]



寿震宇（1993− ），男，宁波大学信息科学与工程学院硕士生，主要研究方向为机器学习、人工智能、大数据检索。

钱江波（1974− ），男，博士，宁波大学信息科学与工程学院教授，主要研究方向为数据处理与挖掘、逻辑电路设计、多维索引与查询优化。

董一鸿（1969− ），男，博士，宁波大学信息科学与工程学院教授，主要研究方向为大数据、数据挖掘和人工智能。

陈华辉（1964− ），男，博士，宁波大学信息科学与工程学院教授，主要研究方向为数据处理与挖掘、云计算。